International Journal of Recent Trends in Engineering & Research
ISSN (ONLINE) : 2455 – 1457
IMPACT FACTOR : 4.101

MAPREDUCE RESEARCH ON WAREHOUSING OF BIG DATA

Dr .A P NIRMALA [1], SHILPA R PATIL[2], SHRUTI N R[3]
[1].Sr.Asst.Prof., Department of MCA, New Horizon College of engineering
[2,3]PG Student, Department of MCA, New Horizon College of engineering

Abstract-The growth of social networks and affordability of various sensing devices has lead to a huge increase of both human and non-human entities that are interconnected via various networks, mostly Internet. All of these entities generate large amounts of various data, and BI analysts have realized that such data contain knowledge that can no longer be ignored. However, traditional support for extraction of knowledge from mostly transactional data - data warehouse - can no longer cope with large amounts of fast incoming various, unstructured data - big data - and is facing a paradigm shift. Big data analytics has become a very active research area in the last few years, as well as the research of underlying data organization that would enhance it, which could be addressed as *big data warehousing*. One research direction is enhancing data warehouse with new paradigms that have proven to be successful at handling big data. Most popular of them is the MapReduce paradigm. This paper provides an overview on research and attempts to incorporate MapReduce with data warehouse in order to empower it for handling of big data.

Keywords- BI analysts, hive, Hbase.

I. INTRODUCTION

Data-oriented systems have recently been faced with overwhelming amount of various-typed data. Data warehouses, core of all traditional decision-support systems, also have to face these challenges. It has become crucial to analyse almost all *possible* data to keep up with the competition by following the shift in data warehousing trends. Transactional data do not provide sufficient support anymore and other emerging data sources must be embraced. These new data sources are part of *big data* phenomena. Traditional data warehouses and RDBMSs are not an adequate storage for big data, and that needs to change because the quantity of unstructured data exceeds the quantity of structured data by four to five times [1]. In big data analytics, MapReduce programming framework has gained popularity because of its ability to effectively process large databases, as well as big data. For that reason, researchers have turned to application of MapReduce to overcome the gap between data warehouses and big data.

Research approaches include integration of data storage systems, changing relational paradigm to non-relational, and new algorithms and approaches to creating and storing data cubes. This paper presents an overview of some of the interesting work done in this field. The paper is organized as follows: section II presents the current data trends, features, storages and paradigms; section III

Presents main issues of data warehouses with big data and unsuitability of Hadoop alone for solving them; section IV presents two research approaches to solving problems described in previous section through describing representative work that has been done; section V discusses the presented approaches; and the section VI concludes this paper.

II. DATA TRENDS

A. Big data

The term *big data* was invented to describe new types of data that emerged with popularization of social networks and data-generating smart or simple devices. There are quite a few big data issues from analytical point of view: they may be unstructured and/or semi-structured and lack formal model and metadata; they arrive fast and in large quantities; they may be incorrect to a certain level. Most importantly, it is questionable if a certain set of big data holds any value at all, but that can only be known through data exploration. Big data features are usually described by *4Vs* – *volume*, *variety*, *velocity*, and *veracity*– But, three more *V* features have been identified [2] variability (in ways to interpret same data), *value* (actual usefulness of extracted information), and *visualization* (need for illustrative visualization of big data sets to understand them).

B. Shared-nothing architecture

Shared-nothing architecture is a distributed computing architecture that features a collection of independent and self-sufficient nodes. These nodes may be various (virtual) machines, where each one has a local disc and main memory. The nodes are connected via network and there is no contention point in it. The phrase *shared-nothing* came from the fact that nodes do not share any kind of memory. The most popular paradigm based on this concept is the MapReduce.

C. MapReduce

MapReduce (MR) has recently become one of the most popular frameworks for processing large amounts of data. It is specially designed for scalability of data processing in the cloud environments. It hides the implementation of its main features from the programmer, e.g. fault tolerance, and load balancing, parallelization and data distribution.

D. NoSQL

The term NoSQL is interpreted as "Not Only SQL". In this case, having *SQL* as a synonym for relational databases, *NoSQL* does not imply the complete absence of *relational*. NoSQL databases are usually open-source and their main features are: data and processing distribution, horizontal scalability, fault tolerance, fast data access, simple APIs, relaxed consistency and being schema-free [3] However, they also have their downsides. The current issue with NoSQL databases is that they do not offer a declarative query language similar to SQL and there is no single, unified model of NoSQL databases.

There are multiple NoSQL database families, which differ significantly in their features. Main NoSQL database families (sometimes referred to as 4 *true* NoSQL database families) are: column-oriented, document-oriented, key-value stores, and graph. Also, there are less commonly mentioned families that had been present long before NoSQL term was used: object-oriented, XML-native, and other database families like multimodal, multivalve, and multidimensional databases.

In attempts of big data warehousing, mostly used NoSQL systems were based on Hadoop’s implementation of MR, running on top of it or along with it, e.g. HBase, Hive, Pig, Spark, Sqoop, Impala, etc., but attempts have also been made without using a MR-based system. Those that best served for research purposes are briefly described below.

1. HBase is a column-oriented distributed database that runs on top of Hadoop. It was modelled after Google BigTable. It features data compression and Bloom filters on column-level. Its tables can be used as input and output for MR. One of the most popular data-driven websites that uses HBase is Facebook, which uses it for its messaging platform.

2. Hive [4] is a data warehouse infrastructure running on top of Hadoop. In Hive, data can be stored in various databases and file systems, all interconnected with Hadoop. It supports SQL-like commands in its language, *HiveQL*, which shifts the focus from low-level programming to a much more declarative level. Given queries are translated into MR jobs that are executed on Hadoop. Hive contains a system catalogue called *metastore* that stores metadata about tables, and that makes it a *data warehouse*.

The idea behind Hive was to bring in well-known concepts of *structuredness* (e.g. tables, columns, partitions and subset of SQL) into unstructured Hadoop environment, but still keeping the flexibility and extensibility Hadoop has. Organization-wise, table is a one logical unit of data and is stored as a directory in HDFS; table partitions are stored as a subdirectory within table's directory; buckets are stored as files within partition's or table's directory, depending on table being partitioned or not.

Hive adopted one of the main changes in approaches to data (warehouse) modelling: *schema-on-read* approach rather than *schema-on-write*, meaning that the data are loaded more quickly because there is no validation against a pre-defined schema. The validation is performed during run-time, which impacts the query time to be slower.
One of the main downsides of Hive's design is its naive simple rule-based optimizer. The authors predicted optimizer improvements as the future research work, as well as exploring other data placement ways, e.g. column-oriented to improve data reading performance.

1. Pig is a high-level platform that runs on top of Hadoop and serves for creating programs running on it. Unlike Hive, which is meant for exploring datasets, Pig is more appropriate for processing data flows [4]. Its language *Pig Latin* is more declarative comparing to MR programming in Java and also supports user-defined functions, written in Python, Java, etc. Pig Hadoop jobs cannot only be run in MR, but also in other frameworks, e.g. Spark.

2. Google BigTable [5] is a NoSQL database for unstructured data. It was built on a few Google technologies, including Google File System. In BigTable, data are mapped by triplets <*row key, column key, timestamp*> and each stored value is an uninterpreted array of bytes, while the timestamps are used for versioning. There is a false perception that BigTable is a column-store database, but its connection to column-oriented approaches lies in usage of *column families*. Column families are groups of column keys that are grouped into sets, i.e. columns are clustered by key. Usually, a column family contains data of the same type.

3. Apache Spark is a cluster-computing framework featuring data parallelism and fault tolerance. It was made to overcome MapReduce's dataflow limitations and has proved to be faster than MR framework. For that reason it should be considered at the time of choosing the right technologies to deal with big data in general, as well as in OLAP.

III. DATA WAREHOUSE'S *BIG DATA ISSUE*

There is a misconception that standard data warehouse architecture is not completely compatible with big data. One of the main arguments of this misconception is the notion that data warehouses are primarily focused on "stock" or fixed data supply oriented on the past and not suited for real time streaming analytics[2] , but real-time BI, which is supported by real-time data warehouses is its counterargument. The problem lies in the intensity of the data streaming, which has become too high with the emergence of big data. The hype around big data has even caused some vendors and analysts to foretell the death of data warehousing and even relational databases. Most popular BI platforms do not support integration with NoSQL databases, but some provide native access to them (e.g. Hive).

MapReduce is considered "appropriate to serve long running queries in batch mode over raw data", which is exactly what ETL process does. However, integrating or replacing traditional systems with Hadoop is a challenging research topic. Despite Hadoop's ability to provide instant access to large amounts of high quality data, the implementation of ETL processes in Hadoop is not simple. The users need to manually write custom programs, which are sometimes not effective and cannot compare to ETL tools for data cleaning, transformation and coordination. Authors in [6] suggest that using ETL Tools is still suitable for small amounts of data, while design of custom programs in Hadoop is more feasible for larger amounts. Another important feature Hadoop lacks is adding dynamic indexes to the framework, which is still an open issue.

IV.MAPREDUCE APPROACHES IN RESEARCH

Research of MapReduce approaches in data warehousing can be grouped into two categories:

i. system-level research: focused on system-level improvements and/or integration;
ii. Data cube research: focused on an OLAP data cube component of data warehouse.

A. System-level research

1) ***Simple data warehouse augmentation***

The simplest way of data warehouse's coexistence with *new* big data solutions was presented in [2], where big data are treated as an additional data source for the data warehouse. This idea does not include integration of data warehouse and big data system on any level, but the data sequentially going through processing systems based on their kind, and in the end - ending up in the data warehouse. Before loading, big data are processed (analysed) in a MR framework, while the rest go through standard cleaning and transformation processes. In this case, the resulting data from the MR processing enrich dimension tables by adding more attributes to them and thus enrich the overall analytical power of the data contained in the data warehouse. Authors pointed out that the good approach would be to leave the original data source intact to enable different types of analyses because the same data processed in different ways can yield different results. However, the issue with this solution is that the programmer or analyst needs skills and knowledge to write map and reduce functions, which may have a big span of complexity – from simple counting to using data mining algorithms and machine learning.

2) ***Hadoop DB***

HadoopDB [7] is a hybrid of parallel databases and MR. It is an open-source solution made of other open-source technologies like Hive, Hadoop and PostgreSQL, all organized in a shared-nothing architecture. The idea behind such integration was to benefit from scalability advantages of MR, and from performance and efficiency featured in parallel databases. Comparing its used storage systems with those used in Hive, HadoopDB uses relational databases in each of the nodes instead of distributed file systems. MR is used as communication layer; Hive serves as a translation layer (queries expressed in SQL are translated into MR jobs); on the database layer there can be multiple nodes running DBMS. The main research contribution was improving query optimization compared to optimization in Hive by pushing query processing logic maximally down to the databases (because Hive did not consider table data collocation in a single node, while HadoopBP does), and modifying Hive by adding four more components to it – *database connector* (for connecting to databases, executing queries and returning the result), *catalog* (for maintaining information about the databases), *data loader* (for Partitioning and re-partitioning of data and data loading), and *SQL-to-MapReduce-to-SQL planner* (for extending Hive query planner). So far, HadoopDB implemented filter, projection and aggregation operators, while the authors expect that this system will benefit from future Hadoop improvements, despite of parallel databases currently still being faster.

3) ***Starfish***

Starfish [8] is a self-tuning system for Hadoop for big data analytics: it tunes Hadoop parameters, but it does not improve its peak performance. Starfish is placed between Hadoop and MR job-submitting clients like Pig, Hive, command line interface, etc. MR jobs, workflows, or collections of workflows are expressed in language *Last word* (Language for Starfish Workloads and Data). The main focus of this system is its adaptation to user's and system's needs. Reasoning behind self-tuning approach lies in Hadoop's suboptimal usage of resources and time. The problem with Hadoop is that there might be around 190 configuration parameters controlling its behaviour, and most of them need to be set manually, e.g. actions like adding and removing a node based on the needs, or rebalance of data. Even for a single MR Job, multiple parameters need to be set, e.g. number of mappers and reducers, controls for network usage, memory allocation settings. Higher-level languages for defining MR jobs – like HiveQL and PigLatin – do not consider cost-based optimization during the query compilation into MR jobs, but only rely on naive hard-coded rules or hints specified by the users. The basic endeavour is to *move the computation to the data* as much as possible to achieve minimal data movement. Starfish provides automatic system tuning throughout the whole data lifecycle: it is able to perform tuning at different levels of granularity – workload, workflow, and job-level – and the interaction of these components provides Starfish's tuning in whole.

4) ***Olap4cloud***

Olap4cloud [1] is an open-source experimental OLAP engine built on top of Hadoop and HBase, designed to efficiently perform queries containing aggregations and grouping on large data sets, and to be horizontally linearly scalable. Its main features include data defragmentation, indexing, and preaggregations. Olap4cloud stores rows with the same dimensions close to one another. For preaggregations, aggregation cuboids are constructed using classic lattice modes, which help to achieve lower latency during execution of queries that use already calculated aggregations. For now, Olap4cloud is still in an early experimental stage: it is able to manage and query cubes (fact tables containing dimensions and measures), but it does not fully support dimension tables, hierarchies and measures: dimensions can only be Java data type *long*, and measures must be in Java data type *double*.

5) ***HBase Lattice***

HBase Lattice [9] is an open-source solution that attempted to provide BI "OLAP-ish" solution based on HBase. Author argues that, in discussion of "*Cassandra is for OLTP, HBase is OLAP* mantra", this system, based on HBase, has limited support for OLAP features: it does not directly support cube models and has no concepts of dimensions, hierarchies, measures and facts, nor has its own query language. It implements incremental OLAP-like cubes: by pre-building cuboids in a cube lattice model, it is able to cope with aggregate queries.

6) *HaoLap*

HaoLap (Hadoop based OLAP) is an OLAP system designed for big data that has some common features with MOLAP tools. It uses simplified multidimensional model for mapping of dimensions and measures, and most importantly - it supports dimension operators (e.g. roll-up). In this system, optimization and speed-up are not achieved using indexes nor pre-computations, but by sharing (horizontal partitioning) and chunk selection. Instead of storing large multidimensional arrays, *calculations* are stored and that simplifies the dimension by having low storage cost and makes this system's OLAP efficient.

Dimension and cube metadata are stored in XML format in a metadata server, while measures are stored in HDFS. Data cubes do not have to be instantiated in memory - OLAP algorithm and MR are used for that. MR jobs are consisted of four parts (input-formatter, mapper, reducer and output-formatter) and are specified as such quadruplets. MR Job operates on the level of chunks, and at the end of the processing chunk is stored in the created chunk files, which are then logically combined into the result – a cube.

7) Octopus

Octopus [11] is a hybrid data processing engine. It is a middleware engine that integrates various backend systems and serves as a one access point (Figure 1.) Between the user and them. The reasoning behind building such integration system is that neither traditional warehousing systems, nor recent big data systems are able to fully leverage the power of the other – neither can fully take over the other's role.

Data are distributed in multiple locations of the system, while the system itself bases its optimization on minimizing the amount of data movement by making a query push-down, which is based on the data's residence. Octopus takes a given query task in an SQL-like form from the user and divides it into multiple subqueries. The Initial query is processed with system components, *parser* and *optimizer*; parser produces a logical operator tree and optimizer produces an optimal set of query plans. Subqueries derived from the initial query are pushed down to the backend systems and processed inside of them. By sending the queries to systems where the data are physically stored and optimally processed, data movement costs are reduced.

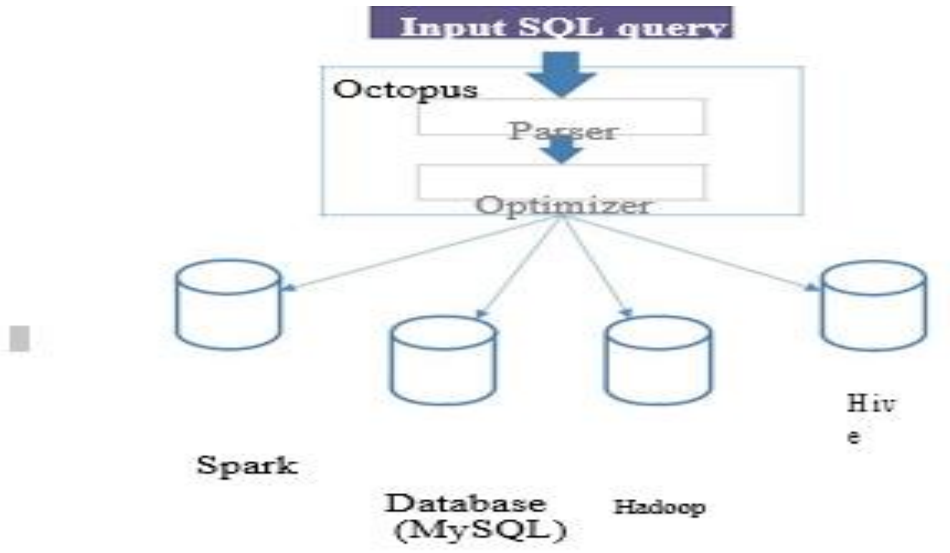

Figure 1. Octopus system overview

Authors have compared Octopus to Spark, because Spark also supports integration of external data sources and implements unnecessary data filtering for reducing data migration. Experiments proved Octopus to be faster than Spark in processing aggregation queries. Possible future development of Octopus lies in adding support for more types of NoSQL databases, as well as integrating more Spark modules, e.g. machine learning.

B. Data cube research

1) Approach by Abello, Ferrarons and Romero

An approach to building data cubes on BigTable was presented in [1]. BigTable is used for storing all the data together temporarily - data are just partially cleaned and integrated before they are stored - without having a concrete schema, so the schema may evolve easily over time. Also, the authors argue that column storage is good enough for data marts because they have well-defined columns, but for the entire data warehouse more flexibility is required because of unclear data structure and continuous schema evolution.

The basic idea is to have a non-formatted chunk of data just stored and associated to the key, which is later, once it is needed, retrieved and used according to the specific needs. That is the opposite from putting a lot of resources and effort into formatting and structuring the data according to the strict star schema, which after data import may not be used for a couple of months, or may even not be used at all. Upon having an analysis to perform, data quality is specified, suitable technology is selected (e.g. ROLAP or MOLAP), and then ETL is used to prepare the data according to needs. On demand, specific and customized data cubes are extracted and stored in a data mart, which may or may not reside in cloud.

For building cubes three typical table access approaches (assuming denormalized data) have been applied in a way they could benefit from MR and BigTable: full scan, index access, and range index scan.

2) MRDataCube

A new cube computation algorithm *MRDataCube* (MapReduce Data Cube) was presented in. MRDataCube is a large-scale data cube computation algorithm that uses MR framework and uses concepts of cuboids and cells. *Cuboid* is considered an aggregated result of aggregate operations upon a cube with the goal of extracting each combination of attributes. *Cells* construct cuboids, and they represent the values stored in them and can be represented as *m*-membered tuple, where *m* is aggregation measurement of the cell.

This algorithm computes the cube in two phases, *MRSpread* and *MRAssemble*. Its main feature is continuous data reduction through these two phases. The First phase produces partial cuboids cells, where one Cuboid cell values may spread through outputs of several Reduce functions. In the second phase, complete cuboids are generated by computing all partial cells, i.e. merging partial cuboids into one cell. Between two phases, result is stored in HDFS. This algorithm showed 2-16 times better performance compared to previous work in that area, algorithms *MRNaive*, *MR2D*, *MRGBLP* and *MRPipeSort*.

3) CN-cube

An operator *CN-CUBE* (Columnar NoSQL cube) for computing OLAP cube from column-oriented data warehouse was presented and it proved to be faster than Hive. The main technology they used was HBase engine. By implementing CN-CUBE operator, authors have attempted to solve one of the most common obstacles of column-oriented databases – the lack of OLAP operators. The CN-CUBE reduces disk access by using a result on an extraction query as a view – based on attributed needed for the cube computation requested by analysts - and avoids re-accessing the data warehouse for performing different aggregations. Using this approach, authors have reduced input and output flows substantially.

This operator performs in three phases. During the first phase, attributes are identified and necessary columns (most recent values) are extracted into an intermediate result. In second phase, intermediate relation columns with the values are hashed to produce the lists of values' positions. These lists allow some columns to be aggregated separately by each dimension. In the final phase, lists of dimensions' positions of intermediate results are associated through the logical *AND* operator.

4) MC-CUBE

Similarly to [13], [14] presents an aggregation operator MC-CUBE (MapReduce Columnar cube) that uses HBase as main technology, and introduces a different approach to building OLAP cubes from columnar "big data warehouse". In this approach, data are taken at a single, least aggregated level, and other aggregates – at other, higher granularity levels – are computed from it. To perform aggregation, this operator performs *invisible join*, typical in column stores, and extends the aggregation to cover "all possible aggregations at different levels of granularity of the cube".

MC-CUBE operator performs seven MR jobs during four phases. During the first phase, aggregations are produced from the data warehouse at lowest and highest granularity levels – joins between fact and dimension tables are materialized and aggregates are calculated. In the second phase, aggregations are performed of each dimension separately, and in the third, rest of the aggregates composing the cube are computed. The final phase replaces dimensions' identifiers with dimensions' attributes, and completes the cube.

V. DISCUSSION

For big data warehousing solution, system integration approaches usually include integration of one or more MR-based technologies like Hive and Hadoop, and combine them with either NoSQL and/or traditional relational storage. Particularly interesting combination is the integration of MapReduce and parallel databases, where the best features are taken from both of them and combined complementarily because one does well what the other does not.

One of the suggested scenarios [3] for big data analysis is to use Hive in Hadoop environment for OLTP, and ROLAP tool for data analysis. However, problems may occur in case when data arrive too quickly and are stored in real-time in Hive, because the existing aggregates in RDBMSs used for OLAP could in a very short time become inaccurate. Other problems are related to the volume of the data; when using partial aggregates, it still may affect the speed OLAP browsing in a negative way. The solution may lie in using MapReduce processing on both OLTP and OLAP side.

In a few works [8], [15], [16] an acronym *MAD* appeared, denoting features expected from big data analytics systems: m*agnetism* means that system attracts all kinds of data, regardless of possibly unknown structure, schema, or even partial values; ***a****gility* stands for system's ability to adapt to data evolutions; ***d****epth* means that the system supports analyses that go to bigger depths than just roll-ups and drill-downs. Authors in [8] even went further and described three more features that would *MADDER* analytics systems need to have: *data-lifecycle-awareness*, *elasticity*, and *robustness*. In future data warehouse modifications and research, these features may prove to be a good guiding principle. It is noticeable that some of these features are found in NoSQL databases, but never all of them. This shows that NoSQL technologies are a good candidate for a data warehouse asset, but chances are small that a NoSQL database would ever completely replace data warehouse's relational basis.

Another potential problem that may affect the course of big data analytics and warehousing development is a large portfolio of technologies and implementations, even among open-source products, and even industry has pointed out this as potential problem [17]. The vendors have approached this matter each in their own way, making new products most compatible with their existing ones. Analysing the general situation from the perspective of data warehousing future, the existing products from which data warehouses may eventually benefit - NoSQL databases and Hadoop implementations and tools - still have many downsides in terms of lacking concepts commonly used in data warehouses and relational databases (e.g. indexing, profound optimizer, ability of having OLAP operators, dimensions and hierarchies, etc.). All of these downsides combined together create obstacles that are being worked on by the current research.

Through the research overview presented in this paper, we can see various system integration approaches [2], [7], [8], [9], [10], [11] to overcoming these issues. System-level integration raises questions of middleware design, data localization, and computation locality i.e. amount data movement, etc. Much work has also been done on optimization issues: it included designing special tuning middleware; building better query and MapReduce job plans; preaggregations – most often in forms of cuboids that are stored in a lattice model; using views to access fewer data – but, one of the most applied approaches is the query push-down technique, i.e. moving the computation to the data, which is guided by the thought that the data should be stored and processed in the system that does suits their nature the best, whether that are RDBMSs, NoSQL databases or MapReduce environment. Possible progress is seen in designing new middleware, either from scratch or by modifying existing systems like Hive.

Presented research overview from the perspective of building data cubes [1], [13], [14] describes representative approaches that have been made with various ways of data storage, especially with column-oriented. Approaches included: storing measures in HDFS while the metadata reside in simple format on server; storing whole data marts in column-oriented [13], [14] storage; collocating data by their type (rows) in column-oriented storage, etc. The diversity is significant, and further research of this matter is expected.

However, one of the most noticeable downsides in some of the presented approaches is considerable simplification – dimensional model is simplified to the extent that the system does not support dimensions and hierarchies, and sometimes even does not differ dimensions from facts. Simplification is an interesting approach, e.g. solving explosive dimension cardinality issue by using some aggregative calculations, but attention should be paid to not over-simplifying the model to the level of its inapplicability for OLAP. To bring big data into the data warehouse, ETL process may need to be loosened to ensure the flexibility for the warehouse and enable the schema to evolve. Still, the main focus remains on creating the much needed OLAP operators.

VI. CONCLUSION

This paper gives an overview of latest research on bringing big data into data warehouses and organization of big data for OLAP analysis using the MapReduce framework. Big data were described by their main features, which are the main reason they cannot be easily let into the strictly structured and highly organized data warehouse environment. However, they are potentially greatly valuable and their share in available data pool is rapidly growing. Thus, they should not be treated as a distinct issue and completely separated from other data, but should be dealt with and made supportive for BI analytics along with traditional transactional data.

NoSQL databases show great potential for handling big data analytics, especially those that implement MapReduce framework. But, there are many issues with these two concepts. The industry has overwhelmed data analytics field with too many implementations that are still not mature enough, mostly in terms of all the underlying supporting features that RDBMSs offer. NoSQL database families differ a lot and may not handle all types of data equally well, and having no unified database model nor does a clear data structure lead to great difficulties in design of analytical systems.

The most realistic future scenario is that the data warehouses will not be replaced by some completely different innovative systems. The more likely scenario is that they will continue to coexist in a symbiotic Relationship with the new, more suitable solutions for big data, where big data would serve as an additional analytical asset.

REFERENCES

I. A. Abelló, J. Ferrarons, and O. Romero, "Building cubes with MapReduce," in Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP '11), Glasgow, Scotland, UK, 2011, pp. 17 – 24.
II. N. Jukić, A. Sharma, S. Nestorov, and B. Jukić, "Augmenting Data Warehouses with Big Data," Information Systems Management, vol. 32, no. 3, pp. 200–209, 2015.
III. J. Duda, "Business Intelligence and NoSQL Databases,"
IV. Information Systems in Management, vol. Vol. 1, no. 1, pp. 25–37, 2012.
V. A. Thusoo et al., "Hive - a petabyte scale data warehouse using
VI. Hadoop," presented at the 2010 IEEE 26th International
VII. Conference on Data Engineering (ICDE), 2010, pp. 996–1005.
VIII. F. Chang et al., "Bigtable: A Distributed Storage System for
IX. Structured Data," ACM Transactions on Computer Systems, vol. 26, no. 2, pp. 1–26, Jun. 2008.
X. T. Šubić, P. Poščić, and D. Jakšić, "Big data in data warehouses," presented at the 4th International Conference the Future of Information Sciences (INFuture2015), Zagreb, Croatia, 2015, pp. 235-244.
XI. A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads," Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 922–933, 2009.
XII. H. Herodotou et al., "Starfish: A Self-tuning System for Big Data Analytics," presented at the 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011), Asilomar, CA, USA, 2011, pp. 261–272.
XIII. D. Lyubimov, "HBase Lattice Quick Start", software manual, source: https://github.com/dlyubimov/HBase-Lattice/
XIV. J. Song, C. Guo, Z. Wang, Y. Zhang, G. Yu, and J.-M. Pierson,
XV. "HaoLap: A Hadoop based OLAP system for big data," Journal of Systems and Software, vol. 102, pp. 167–181, Apr. 2015.
XVI. Y. Chen, C. Xu, W. Rao, H. Min, and G. Su, "Octopus: Hybrid Big Data Integration Engine," presented at the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science, 2015, pp. 462–466.